**Documentação Técnica do Sistema IoT**

**Esta documentação detalha a arquitetura, protocolos e medidas de segurança de um sistema de monitoramento IoT, que utiliza uma abordagem de computação em camadas (Edge, Fog e Cloud) para processamento de dados de sensores.**

**1. Protocolos Utilizados e Justificativas**

**A arquitetura do sistema utiliza uma combinação de protocolos, cada um justificado por seu papel específico no fluxo de dados.**

| **Protocolo** | **Camada de Uso** | **Justificativa** |
| --- | --- | --- |
| **MQTT** | **Edge → Fog** | **Ideal para dispositivos IoT com recursos limitados. É um protocolo leve, com baixo consumo de banda e bateria. Funciona com um modelo de publicação/inscrição, permitindo que o backend receba mensagens de múltiplos sensores de forma eficiente.** |
| **CoAP** | **Edge → Fog** | **Alternativa ao MQTT, projetada para redes de baixo consumo e com perda de pacotes. É utilizado para o sensor de gás, que envia dados através de um método PUT.** |
| **WebSocket** | **Fog → Cloud/Frontend** | **Fornece uma comunicação bidirecional e persistente entre o backend e o navegador. É o protocolo ideal para enviar atualizações em tempo real para o frontend, como a visualização de gráficos dinâmicos com os dados dos sensores.** |
| **HTTP** | **Fog → Cloud/Frontend** | **Utilizado para requisições de API tradicionais, como a obtenção de um token JWT para autenticação no WebSocket. O endpoint /token é uma API RESTful padrão, servida pelo FastAPI.** |

**Exportar para as Planilhas**

**2. Explicação das Camadas Edge, Fog e Cloud**

**A arquitetura do sistema segue um modelo de camadas para otimizar o processamento e a comunicação dos dados.**

* **Camada Edge (Dispositivos Finais): Consiste nos próprios sensores IoT, como o sensor\_temperatura\_mqtt.py e o sensor\_gas\_coap.py. Nesta camada, a coleta e a criptografia dos dados são realizadas diretamente no dispositivo. O processamento é mínimo, focado apenas no que é necessário antes do envio dos dados.**
* **Camada Fog (Nó Intermediário): Representada pelo backend app.py (FastAPI). O backend atua como um "nó de borda", conectando-se aos dispositivos (MQTT Broker, CoAP Server) e aos usuários finais (WebSocket). Nesta camada, ocorrem as seguintes tarefas:**
  + **Recebimento e descriptografia dos dados de diferentes protocolos (MQTT e CoAP).**
  + **Análise preliminar dos dados, como a verificação de anomalias.**
  + **Reempacotamento e encaminhamento dos dados para o frontend via WebSocket.**
* **Camada Cloud (Serviço de Visualização): Corresponde ao frontend (navegador web), que hospeda os arquivos index.html, gas.html e script.js. Embora não seja uma "nuvem" no sentido tradicional, ela representa a camada final onde os dados são consumidos e visualizados em tempo real. As tarefas incluem a conexão WebSocket, a renderização de gráficos e a apresentação dos dados ao usuário.**

**3. Segurança e Medidas de Interoperabilidade**

**O sistema foi projetado com a segurança em mente, especialmente na comunicação entre os dispositivos e o backend.**

* **Criptografia (AES-GCM): A segurança dos dados é garantida através da criptografia AES-GCM (Advanced Encryption Standard com Galois/Counter Mode). Tanto o sensor MQTT quanto o sensor CoAP criptografam o payload JSON antes de publicá-lo. O backend é o único componente que possui a chave para descriptografar os dados.**
* **Autenticação (JWT): A conexão WebSocket é protegida por um token JWT (JSON Web Token). O frontend deve primeiro solicitar um token válido através de um endpoint /token. O backend só aceita conexões WebSocket que apresentem um token válido e não expirado, prevenindo acessos não autorizados.**
* **Interoperabilidade: O backend foi implementado para ser agnóstico em relação aos protocolos dos sensores. Ele é capaz de processar dados vindos de sensores MQTT e CoAP (que usam a mesma lógica de criptografia) e unificá-los para um único canal de comunicação com o frontend (WebSocket), demonstrando a capacidade de interoperar com diferentes tipos de dispositivos IoT.**

**4. Considerações sobre Privacidade e Confiabilidade**

**A privacidade e a confiabilidade são elementos cruciais para a usabilidade e segurança de sistemas IoT.**

* **Privacidade: A criptografia de ponta a ponta (AES-GCM) entre o sensor e o backend protege a privacidade dos dados, garantindo que a informação seja ilegível para qualquer intermediário, como o broker MQTT. Isso impede que terceiros mal-intencionados interceptem e leiam o conteúdo das mensagens.**
* **Confiabilidade: A confiabilidade é assegurada em várias frentes:**
  + **Qualidade de Serviço (QoS) MQTT: O sensor de temperatura utiliza QoS 1, garantindo que a mensagem seja entregue ao broker pelo menos uma vez, mesmo se a conexão for interrompida.**
  + **Arquitetura Assíncrona: O backend utiliza asyncio e uma thread separada para o cliente MQTT, garantindo que o processamento das mensagens não bloqueie a aplicação web e que o sistema se mantenha responsivo.**
  + **Tolerância a falhas: O frontend trata a desconexão do WebSocket de forma graciosa, permitindo que a aplicação se reconecte ou exiba uma mensagem de erro, aumentando a robustez da interface do usuário.**

**Fluxo Completo do Sistema IoT com MQTT, FastAPI e WebSocket**

Este documento descreve o fluxo de dados em um sistema de Internet das Coisas (IoT) que utiliza MQTT para comunicação entre sensores e um backend, e WebSocket para a visualização em tempo real no frontend.

**1. Sensor IoT (Cliente MQTT)**

A função principal do sensor é coletar dados do ambiente (como temperatura, ruído, etc.) e prepará-los para o envio.

* **Processo de Envio:**
  + O sensor lê os dados do ambiente.
  + Os dados são formatados em um payload JSON, por exemplo: {"sensor": "temperature", "value": 25.3}.
  + Este JSON é criptografado usando AES-GCM (ou outro método seguro) para garantir a integridade e confidencialidade.
  + Os bytes criptografados são convertidos para Base64. Essa conversão é necessária para que os dados possam ser enviados como uma string via MQTT.
  + A mensagem final, já em Base64, é publicada em um **broker MQTT** em um tópico específico (ex: sensors/temperature). A porta de comunicação é a 1883 (TCP).

**2. Broker MQTT**

O broker atua como um intermediário central, recebendo mensagens de diversos sensores e as retransmitindo para os clientes que se inscreveram nos tópicos correspondentes.

* **Detalhes Técnicos:**
  + **Porta TCP Padrão:** 1883
  + **Função:** O backend se inscreve nos tópicos de interesse (ex: sensors/#) para receber todas as mensagens que começam com sensors/.
  + O broker não realiza nenhuma descriptografia; ele apenas encaminha os bytes da mensagem original para os clientes inscritos.

**3. Backend (FastAPI com Thread MQTT)**

O backend é a ponte entre o MQTT e o frontend. Ele recebe as mensagens do broker, descriptografa-as e as envia para o frontend em tempo real.

* **Processo:**
  + O backend inicia um cliente MQTT com um ID exclusivo e se conecta ao broker.
  + Ele se inscreve nos tópicos de interesse (ex: sensors/#).
  + Quando uma mensagem é recebida, a função de callback on\_message é acionada.
  + Nessa função, a mensagem é decodificada de Base64 e, em seguida, descriptografada.
  + Um novo pacote JSON é construído para o frontend, contendo o tópico e o payload descriptografado:

JSON

{

"topic": "sensors/temperature",

"payload": {"sensor":"temperature","value":25.3}

}

* + Este pacote é enviado aos clientes do frontend via WebSocket, utilizando asyncio.run\_coroutine\_threadsafe.

**4. WebSocket no Backend**

O WebSocket fornece um canal de comunicação bidirecional em tempo real entre o backend e o navegador.

* **Detalhes Técnicos:**
  + **Porta do Servidor:** 8000 (compartilhada com o FastAPI).
  + **Endpoint:** /ws
  + **Conexão do Frontend:** const ws = new WebSocket("ws://localhost:8000/ws");
  + Cada conexão de frontend é armazenada, permitindo que o backend envie mensagens a todos os clientes conectados assim que os dados chegam do MQTT.

**5.Conexão HTTP e JWT no Backend**

**5.1Geração do Token:**

O backend possui um endpoint HTTP que gera um **JWT (JSON Web Token)** para autenticar o frontend.

**5.2. Uso do Token pelo Frontend**

O frontend realiza uma requisição HTTP para obter o token antes de se conectar ao WebSocket ou fazer chamadas REST protegidas.

**5.3. Validação do Token no Backend**

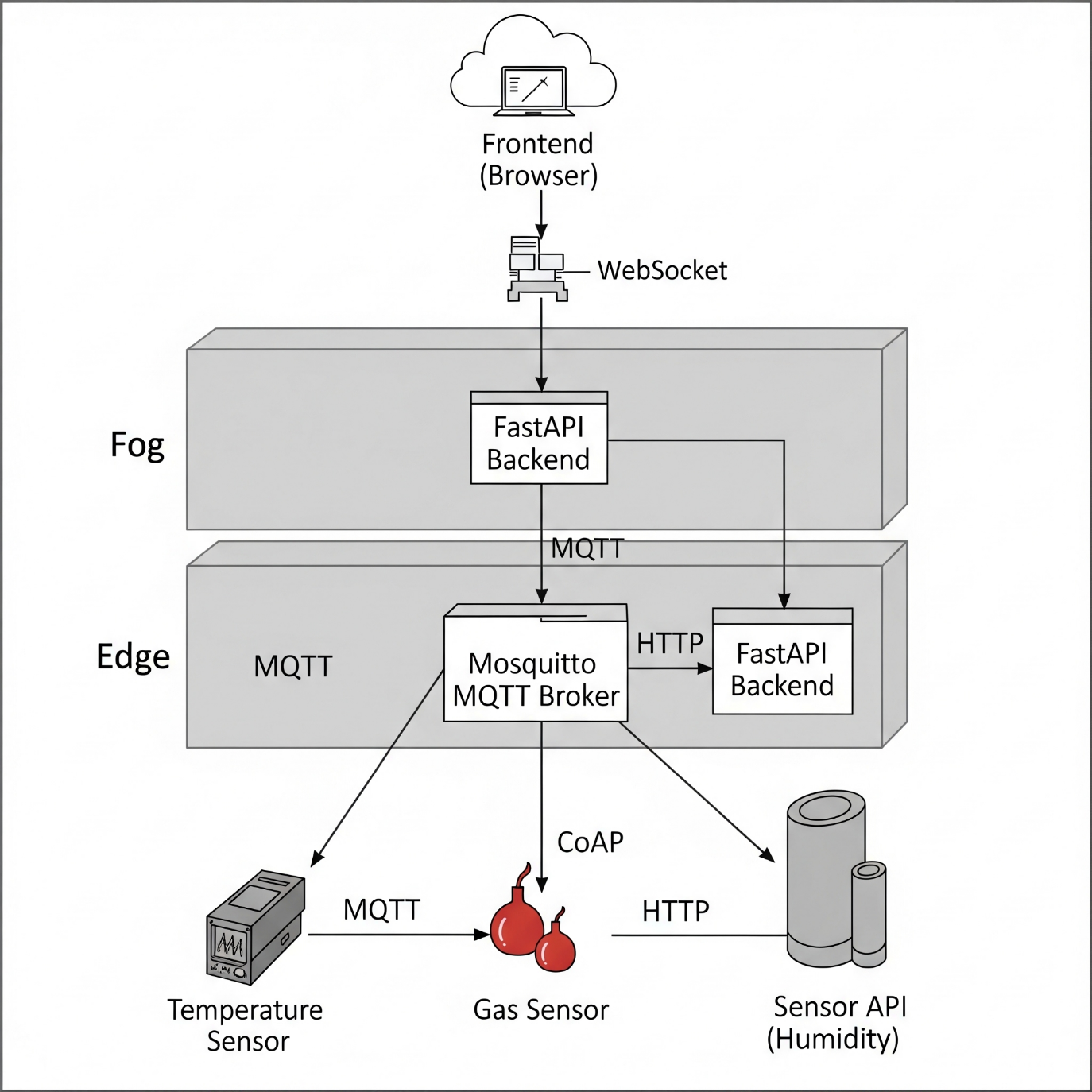
No backend, ao receber requisições HTTP ou conexão WebSocket, o token é validado. Para WebSockets, o token pode ser enviado via query string (?token=...) ou headers. Todas as mensagens do frontend são permitidas somente após validação do token.

**6. Frontend (Navegador)**

O frontend é responsável por receber os dados em tempo real e exibi-los em um formato visualmente agradável, como gráficos.

* **Processo:**
  + O navegador se conecta ao servidor WebSocket no endereço ws://localhost:8000/ws.
  + Ele recebe as mensagens JSON enviadas pelo backend.
  + Com os dados do payload, ele atualiza os gráficos ou outros elementos da interface do usuário (UI), utilizando bibliotecas como Chart.js.
  + O número da porta local do navegador é irrelevante, pois se trata apenas de uma porta de origem temporária para a conexão.

**Resumo do Fluxo e Portas de Comunicação**

O diagrama abaixo ilustra a comunicação entre os componentes e as portas utilizadas.

| Componente | Porta | Protocolo |
| --- | --- | --- |
| Sensor → Broker | 1883 | TCP/MQTT |
| Backend → Broker | 1883 | TCP/MQTT |
| Backend → Frontend | 8000 | WS/WebSocket |
| Navegador | Aleatória (ex: 55471) | WS saída |

**Sensor de Gás (Cenário CoAP + AES)**

**1. Visão Geral**

O sensor de gás monitora os níveis de gases em um ambiente e envia os dados de forma **segura** para o backend usando o protocolo **CoAP (Constrained Application Protocol)**.  
Para garantir confidencialidade, todos os dados enviados pelo sensor são **criptografados com AES-256-GCM** antes de serem transmitidos.

O backend atua como um servidor CoAP, que recebe os dados, descriptografa e os envia em **tempo real** para o frontend via **WebSocket**.

**2. Componentes do Fluxo**

1. **Sensor de Gás (Edge)**
   * Captura leituras de gás periodicamente (ex.: a cada 5 segundos).
   * Formata os dados em JSON, contendo campos como sensor: "gas" e value: <ppm>.
   * Criptografa o JSON usando **AES-256-GCM**, gerando um payload seguro.
   * Envia o payload criptografado para o **backend CoAP** através de requisições **PUT**.
2. **Servidor CoAP (Fog / Backend)**
   * Escuta requisições CoAP na porta definida (ex.: 5683).
   * Ao receber uma mensagem:
     + Descriptografa o payload AES para obter o JSON original.
     + Registra logs com os dados recebidos para monitoramento.
     + Encaminha o pacote para o **sistema de WebSocket**, permitindo que o frontend visualize os dados em tempo real.
   * Responde ao sensor com um código CoAP de sucesso (2.04 Changed) ou erro (4.xx).
3. **Frontend (Visualização)**
   * Conecta ao backend via **WebSocket** para receber atualizações instantâneas.
   * Recebe os dados descriptografados, já convertidos em JSON.
   * Atualiza gráficos ou dashboards em tempo real, mostrando os níveis de gás.

**3. Características e Benefícios**

* **Segurança**: AES-256-GCM garante que os dados transmitidos pelo sensor não possam ser lidos por interceptadores.
* **Eficiência**: CoAP é um protocolo leve, ideal para sensores e dispositivos IoT de baixa capacidade.
* **Interoperabilidade**: Dados padronizados em JSON permitem integração fácil com outros sistemas e APIs.
* **Tempo real**: O backend transmite imediatamente os dados descriptografados via WebSocket, permitindo dashboards ao vivo.
* **Logs detalhados**: Todas as mensagens recebidas são registradas, auxiliando no monitoramento e na auditoria.

**Fluxo do Sensor de Umidade com API Pública**

1. **Coleta de Dados da API Pública**
   * O backend do sensor de umidade realiza requisições HTTP GET periódicas a uma API pública de clima (OpenWeatherMap, por exemplo), solicitando dados de umidade para uma cidade específica.
   * A API retorna um JSON contendo diversos parâmetros meteorológicos, entre eles a umidade relativa do ar.
2. **Processamento e Criptografia (Edge/Fog)**
   * O backend extrai o valor de umidade do JSON retornado pela API.
   * Para garantir a segurança da transmissão, o valor de umidade é criptografado localmente usando **AES-GCM** com uma chave de 256 bits.
   * O processo de criptografia inclui:
     + Geração de um **nonce** (12 bytes aleatórios) para cada payload.
     + Conversão do JSON do sensor ({"sensor": "humidity", "value": valor}) em bytes.
     + Criptografia AES-GCM do payload.
     + Codificação final em Base64 de (nonce + ciphertext), pronta para transmissão.
3. **Distribuição via WebSocket**
   * O backend mantém um endpoint WebSocket dedicado (/ws-umidade).
   * Cada vez que um novo valor de umidade é coletado e criptografado, ele é enviado para todos os clientes conectados via WebSocket.
   * O payload enviado segue o padrão JSON:
   * {
   * "topic": "sensors/humidity",
   * "payload": "<string Base64 do AES>"
   * }
4. **Recepção e Descriptografia no Frontend**
   * O frontend abre uma conexão WebSocket com o backend.
   * Ao receber o payload, ele:
     + Decodifica a string Base64 em bytes.
     + Separa os primeiros 12 bytes como nonce e o restante como ciphertext.
     + Descriptografa o ciphertext usando **AES-GCM** e a mesma chave usada no backend.
     + Converte os bytes de volta em JSON para acessar o valor da umidade.
   * Finalmente, o valor descriptografado é apresentado em tempo real, por exemplo, em gráficos ou dashboards.
5. **Vantagens do Fluxo**
   * **Segurança:** Os dados trafegam criptografados, protegendo informações sensíveis mesmo em redes públicas.
   * **Interoperabilidade:** O backend pode integrar facilmente outros sensores ou APIs externas, padronizando o envio em JSON.
   * **Tempo real:** A comunicação via WebSocket permite atualização imediata dos dados no frontend.
   * **Separação de responsabilidades:** O backend trata da coleta, criptografia e broadcast, enquanto o frontend apenas descriptografa e exibe os dados.